Ann Oper Res (2018) 269:773–790
https://doi.org/10.1007/s10479-017-2556-6

RAOTA-2016

A multi-release software reliability modeling for open source software incorporating dependent fault detection process

Mengmeng Zhu[1] · **Hoang Pham**[1]

Published online: 24 July 2017

Abstract The increasing dependence of our modern society on software systems has driven the development of software products become even more competitive and time-consuming. Single release software product no longer meets the increasing market requirements. Thereby it is important to release multiple version software products in order to add new features in the next release and fix remaining faults from previous release. In this paper, we develop a multi-release software reliability model with consideration of the remaining software faults from previous release and the new introduced-faults (from newly added features). Additionally, dependent fault detection process is taken into account in this research. In particular, the detection of a new fault for developing the next release depends on the detection of the remaining faults from previous release and the detection of the new introduced-faults. The proposed model is validated on the open source software project datasets with multiple releases.

Keywords Software reliability growth modeling · Multi-release software · Dependent fault detection process

Abbreviations

LSE	Least square estimation
MLE	Maximum likelihood estimate
MSE	Mean square error
NHPP	Non-homogeneous Poisson process
OSS	Open source software
PP	Predictive power
PRR	Predictive-ratio risk
SGRM	Software reliability growth model

✉ Hoang Pham
AORpham@gmail.com; hoang84pham@gmail.com

[1] Rutgers University, Piscataway, NJ, USA

Notation

$a(t)$	Total fault content function from previous release
$b(t)$	Total fault content function for newly added features
$d(t)$	Fault detection rate function for the next release
m_0	Expected number of software failures at $t = 0$
$m(t)$	Expected number of software failures by time t
$N(t)$	Total number of software failures in the time interval $[0, t]$
$\lambda(t)$	Failure intensity function $\lambda(t) = d[m(t)]/dt$.

1 Introduction

Software product has been rapidly expanding to a wide array of various real-industry and service-based application. The increasing dependence of our society on software-driven system has led the development of software product becomes very competitive and time-consuming. As the software development moves further away from the rigid and monolithic model, the importance of software multiple release is brought to the vanguard (Saliu and Ruhe 2005). Most of the software organizations release the initial version with sufficient functionalities to meet the customer requirements and occupy a certain portion of market share at first. However, it is unlikely to deliver all features that customers wanted in the single release because of the limited budget, unavailable resource, estimated risk and constrained schedules.

Staying competitive in the market and keep profitable for a software product unlikely happen in this increasing-innovational society if only has a single release especially when rival has a new release carrying more attractive features and satisfying more customer requirements (Saliu and Ruhe 2005). In this point of view, multiple-release planning not only makes software organization easily balance the competing stakeholder's demands and benefits according to the available resources, but lowers the risk of not satisfying customer requirements (Ruhe and Momoh 2005; Svahnberg et al. 2010).

On the other hand, large software system continually needs to align with the changing customer requirements for the sake of market share. In order to get the feedback earlier and figure out what customer really wants, and assigning a lower software development cost, with a certain portion of increments on requirement for multiple release product is essential for the growth of an organization (Maurice et al. 2006; Greer and Ruhe 2004; Missbauer 2002; Mehlawat 2013). Thus, it is plausible for software company to modify the parts of the existing modules to extend the current functionality, usability, and understandability by adding new features and correcting the issues from previous release (Al-Emran and Pfahl 2007; Gorschek and Davis 2008).

Since multi-release is critical for modern software product, release planning is becoming a popular research topic in the past few years. Release planning is a very complex problem. It has to take into account the consequence of feedback and update from customers, the demands of potential customers, market feedback, defects from the previous release and other technical and non-technical constraints (Al-Emran and Pfahl 2007; Ruhe and Momoh 2005). Many researchers have studied software release planning problems (Saliu and Ruhe 2005; Maurice et al. 2006; Al-Emran and Pfahl 2007; Ruhe and Saliu 2005; Svahnberg et al. 2010; Carlshamre et al. 2001; Ho and Ruhe 2013). For instance, Saliu and Ruhe (2005) presented a more comprehensive approach to characterize the relative impact of integrating features into an existing system. They extended the former solution approach EVOLVE*

and proposed S-EVOLVE* method by taking into account the characteristics of the target system for feature implementation. Maurice et al. (2006) proposed F-EVOLVE* approach to determine the desired features based on the financial contribution, which is also the results of extending EVOLVE* approach. Moreover, Ruhe and Saliu (2005) described the principle of "art of release planning" and "science of release planning", respectively. Human intuition, communication and capabilities to negotiate between conflicting objectives and constraints referred to the art of releasing planning; while the science of release planning indicated the emphasizing formalization of the problem and algorithm application optimization.

It is generally considered reliability as a key factor in software quality measurement owing to the fact that it qualifies software failures and misbehaviors (Febrero et al. 2016). As the growth of software size and complexity, how to improve software quality and reliability, control the total cost, and carry adequate and attractive features are critical issues faced by software organization. Therefore, in the past three decades, a great number of Software Reliability Growth Models (SRGMs) has been proposed to quantify the quality of software system. Non-homogeneous Poisson Process (NHPP) is considered as one of the most effective models to study software reliability (Goel and Okumoto 1979; Musa 1975; Huang and Kuo 2002; Kapur et al. 2011; Yamada et al. 1986, 1993; Jeske and Zhang 2005; Zhu and Pham 2016). Nevertheless, most of them only can be applied on a single release. How to model software reliability based on a multiple release perspective just starts gaining researcher's attention not very long.

In this paper, we take into account two types of software faults for developing the next release: (1) *Fault from previous release*, i.e. remaining faults from previous release since it is unlikely to detect and remove all faults within limited resources; (2) *New introduced-faults*, i.e. new features are added in the next release, which also brings new software faults into the next release. We also assume that the detection of software fault for the next release's development depends on the detection of the remaining faults from previous release and the new introduced-faults. To the extent of our knowledge, we haven't seen any research focus on remaining faults from previous release, new introduced-faults and dependent fault detection process in multi-release software reliability modeling.

This paper is organized as follows. In Sect. 2, a literature review is presented. Section 3 outlines the proposed multi-release software reliability modeling. The behaviors of software reliability function illustrated by a theoretical fashion are also discussed. Parameter estimation by the use of Least Square Estimation (LSE) and the comparison criteria are discussed in Sect. 4. Section 5 demonstrates the proposed multi-release software reliability modeling with two practical Open Source Software (OSS) datasets from Apache. Finally, conclusions and future research are described in Sect. 6.

2 Literature review

Most software products are not introduced into the market with full capacities at their initial release, only with sufficient functionalities. New features will be added, and existing features will be enhanced in the next release. A lot of researches have been done for the single version software system for the past few decades. Modeling and predicting software failure behavior are also investigated in those researches. However, most of the existing models developed for single version software product cannot apply on the multi-release software product due to different assumptions and applications. Only a few research studied multi-release software reliability and introduced some models to illustrate software detection process and

fault removal process for multi-release software products. It is thus necessary to investigate software measurement metrics for multi-release software product.

The optimization for software version planning and release has been studied by many researchers (Szöke 2011; Naciri et al. 2015; Etgar et al. 2017; Li et al. 2014. For example, Szöke (2011) proposed a theoretically method for agile release planning. The proposed staged-delivery global optimized model gives the main parameters of the typical agile planning space. Moreover, Naciri et al. (2015) aimed to tackle software release planning problem for a Third Party Application Maintenance (TPM) context. A strategic release planning model of software defects based on Third Party Application Maintenance (TPM) constraints and challenges was proposed to produce an effective release planning of software maintenance in outsourcing context. Etgar et al. (2017) explored several optimization approaches to determine the content and release date for each release in order to provide optimal net present value (NPV). Li et al. (2014) and Mehlawat (2013) proposed a multi-objective multi-choice optimization technique to optimize the requirement choice for the three main objectives, cost, revenue, and uncertainty for robust next release problem.

Given the fact that there was lack of attempt to create a release history database of a large number of projects in the Open source ecosystem, Tsay et al. (2011) created a software release history database including the tools, techniques and pitfalls, to provide insightful and sound information for the future researchers and industrial release engineering practices.

In the real application, Leszak (2005) provided a study of several processes on an industrial large-scale multi-release software system. Sukhwani et al. (2016) performed the software reliability analysis of the flight software of a recent launches space mission. They also linked the activities in the major releases with the problems encountered during the development of those releases. Mahimkar (2016) focused on the detection of software upgrades on smartphones and analyzing their service performance impact, for example, smartphone-centric, or network-centric impact.

In terms of software reliability study, Garmabaki et al. (2011) incorporated different severities level used to describe the difficulty of correcting faults in the upgrade process to develop a multi up-gradation software reliability model. Faults are classified into two categories, simple fault and hard fault. The fault removal for the development of the new release depends on the fault from previous release and fault generated in that release. Hu et al. (2011) considered the effect of multiple releases on the fault detection process in software development. They assumed that there is no gap between the release of previous version and the development of next version. Optimal release time for each version is also present in this paper.

Kapur et al. (2012) introduced the combined effect of schedule pressure and resource limitations by the use of Cobb–Douglas production function in software reliability modeling. The Cobb–Douglas function illustrates the total production output can be obtained by the amount of labor input, capital input, and total factor productivity. An optimal release planning problem is formulated in this paper for software with multiple releases with the solution obtained by applying genetic algorithm method.

Pachauri et al. (2015) proposed a software reliability growth model by considering fault reduction factors (FRFs) and extended this idea to multi-release software systems. FRFs is defined as the ratio of the total number of reduced faults to the total number of failures, which can be affected by other factors, such as resources allocation, debugging time lag and imperfect debugging.

Yang et al. (2016) incorporated fault detection and fault correction process in multi-release software reliability modeling. There is a time delay in fault repair after detecting faults. The time delay function is explained by an exponential function or a gamma function. They also

assumed the faults in a new version including both undetected faults from last version and new introduced faults during the development process of the new version.

Ahmadi et al. (2016) incorporated bugs removed from pre-commit test and bugs reported by parallel debugging test based on software lifecycle development process (SDLC) proposed by Jørgensen (2001). Additionally, the fault removal of the new release depends on the reported faults from previous release and the faults generated by the new functionalities.

However, most literature aimed to develop multi-release software reliability model only by optimizing software cost model to determine software release time except Yang et al. (2016). Some of them have also incorporated the fault removal of the next release depends on the reported faults from previous release and the faults generated by the new adding functionalities, like Ahmadi et al. (2016), but they didn't consider the dependent fault detection process in the next release. Hence, our research focuses on the dependent fault detection process for next release's development along with the consideration of the remaining faults from previous release and the new introduced-faults (from newly added features); in other words, the detection of the new faults for the development of next release depends on the detection of remaining faults from previous release and the new introduced-faults.

3 The multi-release software reliability modeling framework

3.1 Multi-release software reliability modeling

It is unlikely to get bug-free software product within limited resources and tightened schedules. Software detection process still follows an NHPP process for developing the next release. The cumulative number of detected faults $N(t)$ follows Poisson Process.

$$\Pr\{N(t) = n\} = \frac{(m(t))^n \exp(-m(t))}{n!}, \quad \text{for } n = 0, 1, 2, \ldots$$

where $m(t)$ is the mean value function of the counting process $N(t)$.

Two types of software faults will be addressed in this paper. Remaining faults from previous release (Part I) and new introduced-faults (Part II) will be both incorporated with the aim of developing the next release. Fault detection is a dependent process. We assume the detection of a software fault depends on the fault detected from Part I and Part II.

Thus, the multi-release software reliability modeling can be formulated as follows:

$$\frac{dm(t)}{dt} = d(t)[a(t) - m(t)][b(t) - m(t)]m(t) \tag{1}$$

where $m(t)$ represents the expected number of software failures by time t, $d(t)$ denotes the fault detection rate function, a(t) and b(t) represent the total remaining faults from previous release and the total fault content of the current release, respectively. In this paper, we assume that

$$a(t) = a, b(t) = b \tag{2}$$

Substitute (2) into (1), we can obtain a general solution for the mean value function m(t) by solving the following equation:

$$e^{dt+C_0} = m(t)^{\frac{1}{ab}} (m(t) - a)^{\frac{1}{a(a-b)}} (m(t) - b)^{-\frac{1}{b(a-b)}} \tag{3}$$

where C_0 is a constant. In this study, we consider that the initial solution of the function m(t) is as follows:

$$m\,(t=0)=m_0 \tag{4}$$

where $m_0 \geq 0$ is unknown. At time $t = 0$, the expected number of initial software failures is m_0. Since multiple software releases are considered in this study, the expected number of software failures at the beginning of next release should be less than or equal to the expected number of failures at the end of previous release. Additionally, it is unlikely to remove all the software faults for each release due to the limitation of all available resource, including software programmer's domain knowledge and other environmental factors, as seen in reference Al-Emran and Pfahl (2007), Ruhe and Momoh (2005), Gorschek and Davis (2008), Kapur et al. (2012), Yang et al. (2016), Pachauri et al. (2015), Zhu et al. (2015).

Substitute (4) into (3), we obtain

$$e^{C_0}=m_0^{\frac{1}{ab}}\,(m_0-a)^{\frac{1}{a(a-b)}}\,(m_0-b)^{-\frac{1}{b(a-b)}} \tag{5}$$

Thus, the solution for the function m(t) from the Eq. (1) can be obtained by solving the following equation:

$$e^{dt}=\left(\frac{m\,(t)}{m_0}\right)^{\frac{1}{ab}}\left(\frac{m\,(t)-a}{m_0-a}\right)^{\frac{1}{a(a-b)}}\left(\frac{m\,(t)-b}{m_0-b}\right)^{-\frac{1}{b(a-b)}} \tag{6}$$

3.2 Multi-release software reliability function discussion

Let

$$g\,(t)=e^{dt+C_0} \tag{7}$$

and

$$f\,(x)=x^{\frac{1}{ab}}\,(x-a)^{\frac{1}{a(a-b)}}\,(x-b)^{-\frac{1}{b(a-b)}} \tag{8}$$

We now present the following main results.

Lemma 1 *The solution $m(t)$ of Eq. (3) can be obtained by solving the following function*

$$g\,(t)=f\,(m\,(t)) \tag{9}$$

and,

- *If $g\,(0) > \max\{f\,(m\,(t))\}$ then there exists no solution, as shown in Fig. 1a.*
- *Otherwise, there exists at least one solution for the function $m(t)$, as shown in Fig. 1b.*

where the function g(t) and $f\,(x)$ are given in Eqs. (7) and (8), respectively.

The proof of Lemma 1 is in the Appendix.

4 Parameter estimation and comparison criteria

4.1 Parameter estimation

Most SRGMs use the least square estimate (LSE) or maximum likelihood estimate (MLE) to estimate the parameters carried in the model. For example, minimizing the Eq. (10) or maximizing the Eq. (11).

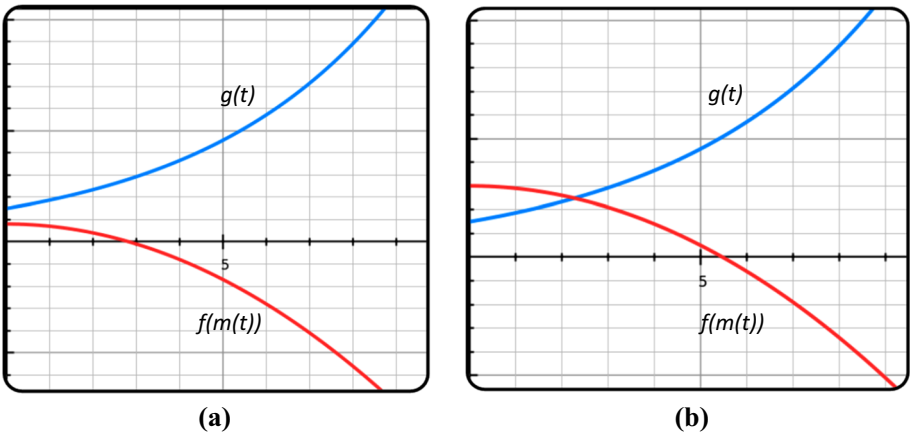

Fig. 1 Illustration of solutions for convex function

$$f\,(t) = \sum_{i=1}^{n} \left(m\,(t_i) - y_i\right)^2 \tag{10}$$

$$LLF = \sum_{i=1}^{n} (y_i - y_{i-1}) \log\left[m\,(t_i) - m\,(t_{i-1})\right] - m\,(t_n) - \sum_{i=1}^{n} \log\,(y_i - y_{i-1})! \tag{11}$$

In this paper, we apply LSE to minimize Eq. (12) to estimate the parameters. Since $g\,(t) = f\,(m\,(t))$, indeed, $log\,[g\,(t)] = log\,[f\,(m\,(t))]$. The optimization function is given by

$$\min S\,(a, b, d) = \sum\nolimits_{a,b,d} (\log\,[f\,(y_i)] - \log\,[[g\,(t_i)]])^2 \tag{12}$$

where y_i is the observed number of failures at time t_i, $g\,(t_i) = e^{dt_i + C_0}$. The Lemma 1 presented in Sect. 3 is to show that there exists the solutions and explain the behaviors of the proposed model. The Genetic Algorithm (GA) is employed in order to solve the optimization function as given in Eq. 12. The schematic diagram of the algorithm (Kachitvichyanukul 2012) is described in Fig. 2. We use Matlab Optimization Toolbox to solve the optimization function and estimate parameters.

4.2 Comparison criteria

(1) Mean squared error

$$MSE = \frac{\sum_{i=1}^{n} \left(\hat{m}\,(t_i) - y_i\right)^2}{n - N} \tag{13}$$

The mean squared error (MSE) measures the distance of a model estimate from the observed data where n is the total number of observations and N represents the number of unknown parameters in each model.

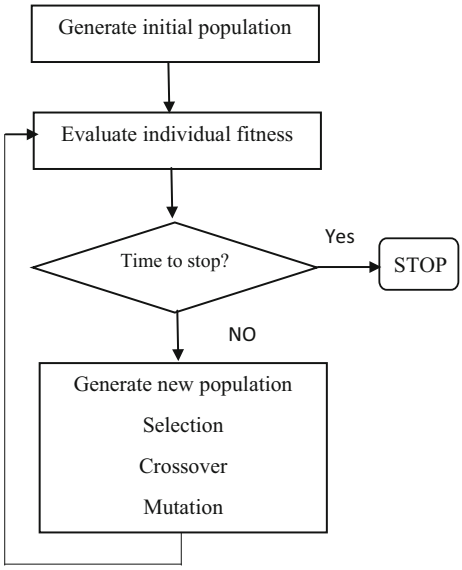

Fig. 2 Schematic diagram of Generic Algorithm

(2) Predictive-ratio risk and predictive power (Pham 2014)

$$PRR = \sum_{i=1}^{n} \left(\frac{\hat{m}(t_i) - y_i}{\hat{m}(t_i)} \right)^2 \tag{14}$$

$$PP = \sum_{i=1}^{n} \left(\frac{\hat{m}(t_i) - y_i}{y_i} \right)^2 \tag{15}$$

The predictive-ratio risk (PRR) and the predictive power (PP) are calculated to compare the power of different models. PRR measures the distance of the model estimates from the actual data against the model estimates; while predictive power (PP) measures the distance of the model estimates from the actual data against the actual data.

(3) Variation

The Variation is defined as (Huang and Kuo 2002):

$$Variation = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(y_i - \hat{m}(t_i) - Bias \right)^2} \tag{16}$$

where

$$Bias = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{m}(t_i) - y_i \right) \tag{17}$$

The smaller the variation, the better of the model should be.

5 Numerical example

In this section, two numerical applications are given to validate the multi-release software reliability model. We employ two datasets both collected from Open Source software (OSS) project. Open Source Software (OSS) is a new way to build a global-based large software

Table 1 Open source software Juddi project data

Week	Cum. failures	Week	Cum. failures
1	10	32	210
2	12	33	210
3	20	34	210
4	31	35	211
5	33	36	213
6	41	37	213
7	47	38	214
8	54	39	217
9	64	40	217
10	74	41	217
11	77	42	218
12	99	43	218
13	110	44	218
14	118	45	218
15	120	46	221
16	127	47	221
17	131	48	221
18	136	49	221
19	137	50	234
20	137	51	249
21	139	52	267
22	144	53	273
23	155	54	279
24	160	55	290
25	160	56	297
26	169	57	314
27	170	58	336
28	180	59	345
29	193	60	387
30	194	61	393
31	195	–	–

system, which differs in many perspectives with the traditional software engineering (Raymond 2001). The evolution process of OSS is much faster than the traditional close source software. Widespread OSS projects bring in a great change in terms of software development paradigms and software architectures (Li et al. 2011; Raymond 2001; Li et al. 2006).

5.1 Juddi project data

Juddi Open Source Software project data is shown in Table 1. Failure dataset from week 1 to week 31 are considered as Release 1; failure dataset from week 32 to week 49 are considered as Release 2; failure dataset from week 50 to week 61 are considered as Release 3. We use Release 2 in this paper to validate our proposed model. The parameter estimate and model comparison are described in Table 2.

Table 2 Parameter estimate and model comparison for Juddi dataset

Model	MVF	MSE	PRR	PP	Variation	Parameter estimate
G-O model	$m(t) = a\left(1 - e^{-bt}\right)$	131.840	0.041	0.046	4.799	$\hat{a} = 227$
						$\hat{b} = 2.42$
Inflection S-shaped model	$m(t) = \frac{a\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$	226.793	0.064	0.074	4.935	$\hat{a} = 230$
						$\hat{b} = 9.07$
						$\hat{\beta} = 830.67$
Delayed S-shaped model	$m(t) = a\left(1 - (1 + bt)\, e^{-bt}\right)$	93.698	0.030	0.033	4.304	$\hat{a} = 225.01$
						$\hat{b} = 4.33$
Yamada imperfect debugging model	$m(t) = a\left[1 - e^{-bt}\right]\left[1 - \frac{\alpha}{b}\right] + \alpha a t$	254.213	0.072	0.083	5.747	$\hat{a} = 231$
						$\hat{b} = 2.18$
						$\hat{\alpha} = 3.64 \times 10^{-5}$
PNZ model	$m(t) = \frac{a\left[\left(1-e^{-bt}\right)\left(1-\frac{\alpha}{b}\right)+\alpha t\right]}{1+\beta e^{-bt}}$	211.036	0.056	0.064	4.787	$\hat{a} = 229$
						$\hat{b} = 6.38$
						$\hat{\alpha} = 1 \times 10^{-6}$
						$\hat{\beta} = 52.69$
Pham–Zhang IFD	$m(t) = a - ae^{-bt} \times \left(1 + (b + d)\, t + bdt^2\right)$	368.638	0.101	0.120	5.601	$\hat{a} = 234$
						$\hat{b} = 9.20$
						$\hat{d} = 98.95$
Dependent-parameter model	$m(t) = \alpha\,(1 + \gamma t) \times \left(\gamma t + e^{-\gamma t} - 1\right)$	23163.48	85316.33	8.153	108.378	$\hat{\alpha} = 364.85$
						$\hat{\gamma} = 0.0625$

Table 2 continued

Model	MVF	MSE	PRR	PP	Variation	Parameter estimate
Proposed model	$e^{dt} = (\frac{m(t)}{m_0})^{\frac{1}{ab}} \times \left(\frac{m(t)-a}{m_0-a}\right)^{\frac{1}{a(a-b)}} \times \left(\frac{m(t)-b}{m_0-b}\right)^{-\frac{1}{b(a-b)}}$	34.467	0.011	0.011	3.322	$\hat{a} = 254.84$ $\hat{b} = 221$ $\hat{m}_0 = 100.00$ $\hat{d} = 2.36 \times 10^{-4}$

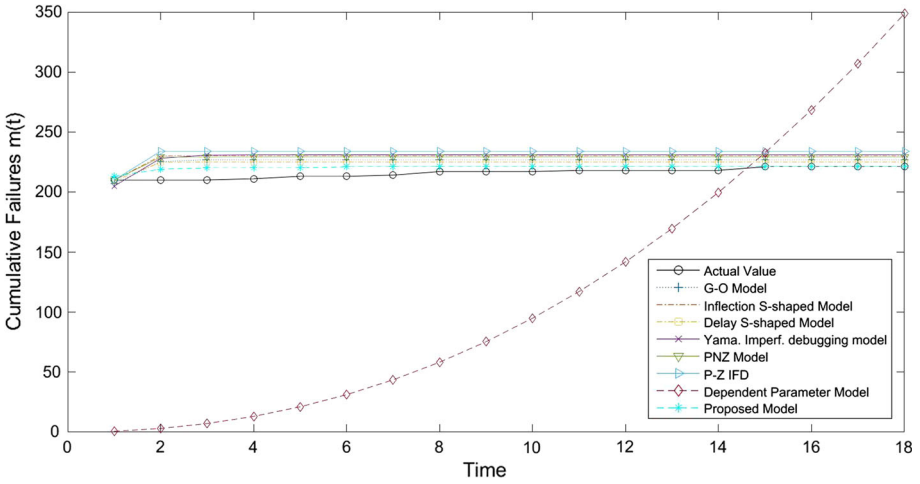

Fig. 3 Comparison between proposed model and other models for Juddi dataset

Table 3 Open source software APACHE 2.0

Day	Cum. failures	Day	Cum. failures
1	1	26	35
2	3	28	36
3	5	29	37
4	8	30	39
5	11	31	40
7	13	32	41
8	14	35	44
9	15	38	45
10	16	39	46
11	17	42	47
15	20	43	48
16	22	49	51
17	25	50	52
18	26	51	53
19	27	57	54
22	30	66	55
23	31	70	56
24	32	81	57
25	34	164	58

As shown in Table 2, we can see that the proposed model has the best performance in terms of all criteria present here. Figure 3 illustrates the comparison between model prediction and observed failure data. All models showing in Table 2 only consider single-release software product except proposed model. In other words, they didn't consider the remaining faults from the previous release since most of models assume all software faults will be removed before software company release the product.

Table 4 Parameter estimate and model comparison for APACHE dataset

Model	MVF	MSE	PRR	PP	Variation	Parameter estimate
G-O model	$m(t) = a\left(1 - e^{-bt}\right)$	38.317	0.169	0.268	8.449	$\hat{a} = 82.99$
						$\hat{b} = 0.022$
Inflection S-shaped model	$m(t) = \frac{a\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$	104.096	2.299	1.304	17.380	$\hat{a} = 75.017$
						$\hat{b} = 0.019$
						$\hat{\beta} = 0.274$
Delayed S-shaped model	$m(t) = a\left(1 - (1+bt)\,e^{-bt}\right)$	21.784	0.230	0.223	4.930	$\hat{a} = 69$
						$\hat{b} = 0.06$
Yamada imperfect debugging model	$m(t) = a\left[1 - e^{-bt}\right]\left[1 - \frac{\alpha}{b}\right] + \alpha at$	26.670	0.117	0.178	5.241	$\hat{a} = 79.69$
						$\hat{b} = 0.021$
						$\hat{\alpha} = 1 \times 10^{-4}$
PNZ model	$m(t) = \frac{a[\left(1-e^{-bt)}\right)\left(1-\frac{\alpha}{b}\right)+\alpha t]}{1+\beta e^{-bt}}$	64.302	1.129	0.754	8.484	$\hat{a} = 65$
						$\hat{b} = 0.121$
						$\hat{\alpha} = 1 \times 10^{-4}$
						$\hat{\beta} = 26.178$
Pham–Zhang IFD	$m(t) = a - ae^{-bt} \times \left(1 + (b+d)\,t + bdt^2\right)$	30.828	0.309	0.300	5.656	$\hat{a} = 71.2$
						$\hat{b} = 0.0575$
						$\hat{d} = 9.15 \times 10^{-5}$

Table 4 continued

Model	MVF	MSE	PRR	PP	Variation	Parameter estimate
Dependent-parameter model	$m(t) = \alpha(1+\gamma t) \times \left(\gamma t + e^{-\gamma t} - 1\right)$	1587.346	3843.289	18.719	71.169	$\hat{\alpha} = 101.11$
						$\hat{\gamma} = 0.007$
Proposed model	$e^{dt} = (\frac{m(t)}{m_0})^{\frac{1}{ab}} \times \left(\frac{m(t)-a}{m_0-a}\right)^{\frac{1}{a(a-b)}} \times \left(\frac{m(t)-b}{m_0-b}\right)^{-\frac{1}{b(a-b)}}$	13.197	0.155	0.171	3.630	$\hat{a} = 90.038$
						$\hat{b} = 61.001$
						$\hat{m}_0 = 20$
						$\hat{d} = 1.165 \times 10^{-5}$

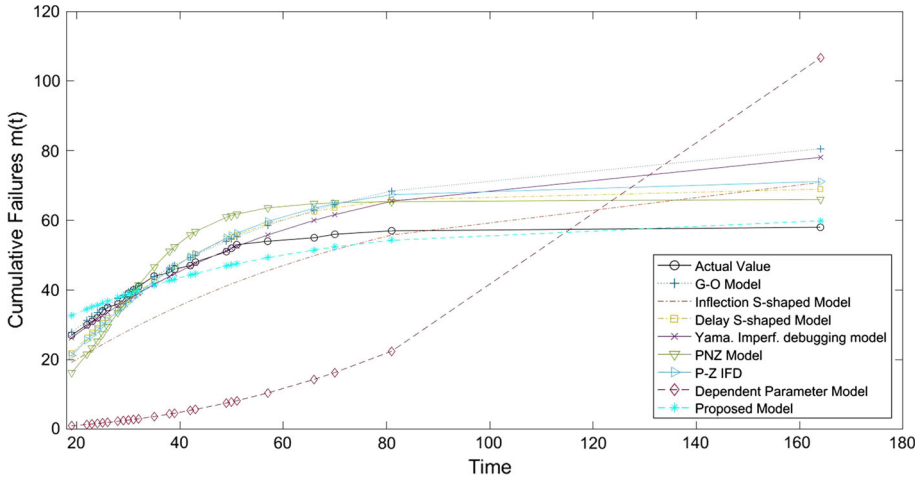

Fig. 4 Comparison between proposed model and other models for Apache 2.0 dataset

5.2 Apache 2.0

Apache 2.0[1] is available on the website since 2002. The first two releases are employed to verify the proposed model, as shown in Table 3. The failure data from day 1 to day 18 is taken into account as Release 1; failure data from day 19 to day 164 is considered as Release 2.

We can see that the proposed model provides the smallest MSE, PP, and Variation as shown in Table 4. The PRR value even though is not the smaller one, however, 0.155 is just slightly higher than 0.117. It is thus considering the proposed model presents the best performance to model this dataset. Figure 4 also plots the comparison between the predicted values and the observed values.

In summary, the proposed model has considered a dependent fault detection process. Specifically, the newly detected faults depend on the detection of remaining faults from previous release and new introduced faults. In order to detect a new fault, we need to detect corresponded faults from remaining faults from previous release and new introduced faults first. Therefore, there is only a small portion of software faults detected for developing the next release.

6 Conclusion remark and future research

It is unlikely to deliver all the features in a single release for the modern software products. The proposed software reliability model provides a new paradigm to integrate the dependent fault detection process and different types of software faults in multiple software release. Due to the resource limitation, there also exists a portion of undetected software faults for the current release. Thus, how to incorporate the faults from previous release into the development for next release becomes an important issue.

As an effort to reflect the development of multi-release software, remaining faults from previous release, new introduced-faults, and dependent fault detection process are discussed

[1] https://www.apache.org/.

in this paper. In order to accurately illustrate the performance of the proposed model, we employ two datasets both collected from Open Source Software (OSS) project to validate the usage of the model. The behavior of software reliability function is studied as well. We are currently investigating the new features adding in the next release and the remaining faults from previous release as fixed numbers in this study, which can be extended as a random number or as a time-dependent function corresponding to its optimal profit and release time for the organization. The impact of environmental factors (Zhu et al. 2015) during the software development process can be considered into the future research as well.

Appendix

Proof of Lemma 1 We need to show that: (a) the function $g(t)$ in Eq. (7) is convex, and (b) the function $f(x)$ in Eq. (8) is concave.

(a) Since d is non-negative and $g''(t) = d^2 e^{dt+C_0} > 0$, thus function $g(t)$ is convex.

(b) $f(x) = x^{\frac{1}{ab}}(x-a)^{\frac{1}{a(a-b)}}(x-b)^{-\frac{1}{b(a-b)}}$

$$
\begin{aligned}
f'(x) &= \frac{1}{ab}x^{\frac{1}{ab}-1}(x-a)^{\frac{1}{a(a-b)}}(x-b)^{-\frac{1}{b(a-b)}} + \frac{1}{a(a-b)}x^{\frac{1}{ab}}(x-a)^{\frac{1}{a(a-b)}-1}(x-b)^{-\frac{1}{b(a-b)}} \\
&\quad - \frac{1}{b(a-b)}x^{\frac{1}{ab}}(x-a)^{\frac{1}{a(a-b)}}(x-b)^{-\frac{1}{b(a-b)}-1} \\
&= x^{\frac{1}{ab}}(x-a)^{\frac{1}{a(a-b)}}(x-b)^{-\frac{1}{b(a-b)}}\left(\frac{1}{abx} + \frac{1}{a(a-b)(x-a)} - \frac{1}{b(a-b)(x-b)}\right) \\
&= x^{\frac{1}{ab}}(x-a)^{\frac{1}{a(a-b)}}(x-b)^{-\frac{1}{b(a-b)}}\frac{(ab+a+b)-(a+b)x}{abx(x-a)(x-b)} \\
&= f(x)\frac{(ab+a+b)-(a+b)x}{abx(x-a)(x-b)} \\
&= [f(x)]^{-1}\frac{(ab+a+b)-(a+b)x}{ab} \\
f''(x) &= -[f(x)]^{-2}f'(x)\frac{(ab+a+b)-(a+b)x}{ab} - [f(x)]^{-1}\frac{a+b}{ab} \\
&= -\frac{f'(x)\left\{[f(x)]^{-1}\frac{(ab+a+b)-(a+b)x}{ab}\right\}}{f(x)} - [f(x)]^{-1}\frac{a+b}{ab} \\
&= -\frac{\left[f'(x)\right]^2}{f(x)} - \frac{\frac{a+b}{ab}}{f(x)} < 0
\end{aligned}
$$

Note that $f(x) = g(t) > 0$ in Eq. (3), $a > 0$ and $b > 0$, therefore, function $f(x)$ in Eq. (8) is concave.

Since the function $g(t)$ is convex, and $f(x)$ is concave, the results in Lemma 1 follow accordingly. □

References

Ahmadi, M., Mahdavi, I., & Garmabaki, A. H. S. (2016). Multi up-gradation reliability model for open source software. In U. Kumar, A. Ahmadi, A. K. Verma, & P. Varde (Eds.), *Current trends in reliability, availability, maintainability and safety* (pp. 691–702). Berlin: Springer.

Al-Emran, A., & Pfahl, D. (2007). Operational planning, re-planning and risk analysis for software releases. In: *International conference on product focused software process improvement* (pp. 315–329). Berlin: Springer.

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., & och Dag, J. N. (2001). An industrial survey of requirements interdependencies in software product release planning. In *Proceedings of the fifth IEEE international symposium onrequirements engineering* (pp. 84–91).

Etgar, R., Gelbard, R., & Cohen, Y. (2017). Optimizing version release dates of research and development long-term processes. *European Journal of Operational Research*, *259*(2), 642–653.

Febrero, F., Calero, C., & Moraga, M. Á. (2016). Software reliability modeling based on ISO/IEC SQuaRE. *Information and Software Technology*, *70*, 18–29.

Garmabaki, A. H., Aggarwal, A. G., & Kapur, P. K. (2011). Multi up-gradation software reliability growth model with faults of different severity. In *2011 IEEE international conference on industrial engineering and engineering management (IEEM)* (pp. 1539–1543).

Goel, A. L., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, *28*(3), 206–211.

Gorschek, T., & Davis, A. M. (2008). Requirements engineering: in search of the dependent variables. *Information and Software Technology*, *50*(1), 67–75.

Greer, D., & Ruhe, G. (2004). Software release planning: An evolutionary and iterative approach. *Information and Software Technology*, *46*(4), 243–253.

Ho, J., & Ruhe, G. (2013). Releasing sooner or later: An optimization approach and its case study evaluation. In *Proceedings of the 1st international workshop on release engineering* (pp. 21–24). IEEE Press.

Hu, Q. P., Peng, R., Xie, M., Ng, S. H., & Levitin, G. (2011). Software reliability modelling and optimization for multi-release software development processes. In *2011 IEEE international conference onindustrial engineering and engineering management* (pp. 1534–1538).

Huang, C. Y., & Kuo, S. Y. (2002). Analysis of incorporating logistic testing-effort function into software reliability modeling. *IEEE Transactions on Reliability*, *51*(3), 261–270.

Jeske, D. R., & Zhang, X. (2005). Some successful approaches to software reliability modeling in industry. *Journal of Systems and Software*, *74*(1), 85–99.

Jørgensen, N. (2001). Putting it all in the trunk: Incremental software development in the FreeBSD open source project. *Information Systems Journal*, *11*(4), 321–336.

Kachitvichyanukul, V. (2012). Comparison of three evolutionary algorithms: GA, PSO, and DE. *Industrial Engineering and Management Systems*, *11*(3), 215–223.

Kapur, P. K., Pham, H., Aggarwal, A. G., & Kaur, G. (2012). Two dimensional multi-release software reliability modeling and optimal release planning. *IEEE Transactions on Reliability*, *61*(3), 758–768.

Kapur, P. K., Pham, H., Anand, S., & Yadav, K. (2011). A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. *IEEE Transactions on Reliability*, *60*(1), 331–340.

Leszak, M. (2005). Software defect analysis of a multi-release telecommunications system. In *International conference on product focused software process improvement* (pp. 98-114). Berlin: Springer.

Li, L., Harman, M., Letier, E., & Zhang, Y. (2014). Robust next release problem: Handling uncertainty during optimization. In *Proceedings of the ACM 2014 annual Conference on Genetic and Evolutionary Computation* (pp. 1247–1254).

Li, X., Li, Y. F., Xie, M., & Ng, S. H. (2011). Reliability analysis and optimal version-updating for open source software. *Information and Software Technology*, *53*(9), 929–936.

Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., & Zhai, C. (2006). Have things changed now?: An empirical study of bug characteristics in modern open source software. In *Proceedings of the ACM 1st workshop on architectural and system support for improving software dependability* (pp. 25–33).

Mahimkar, A. (2016). Detecting and diagnosing performance impact of smartphone software upgrades. In *2016 IEEE 12th international conference on network and service management (CNSM)* (pp. 188–194).

Maurice, S., Ruhe, G., & Saliu, O. (2006). Decision support for value-based software release planning. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, & P. Grünbacher (Eds.), *Value-based software engineering* (pp. 247–261). Berlin: Springer.

Mehlawat, M. K. (2013). A multi-choice goal programming approach for COTS products selection of modular software systems. *International Journal of Reliability, Quality and Safety Engineering*, *20*(6), 1350026-1-18.

Missbauer, H. (2002). Aggregate order release planning for time-varying demand. *International Journal of Production Research*, *40*(3), 699–718.

Musa, J. D. (1975). A theory of software reliability and its application. *IEEE Transactions on Software Engineering*, *3*, 312–327.

Naciri, S., Idrissi, M. A. J., & Kerzazi, N. (2015). A strategic release planning model from TPM point of view. In *2015 10th international conference on IEEE intelligent systems: Theories and applications (SITA)* (pp. 1–9).

Pachauri, B., Dhar, J., & Kumar, A. (2015). Incorporating inflection S-shaped fault reduction factor to enhance software reliability growth. *Applied Mathematical Modelling*, *39*(5), 1463–1469.

Pham, H. (2014). A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments. *Optimization - A Journal of Mathematical Programming and Operations Research*, *63*(10), 1481–1490.

Raymond, E. S. (2001). *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary*. Sebastopol: O'Reilly Media Inc.

Ruhe, G., & Momoh, J. (2005). Strategic release planning and evaluation of operational feasibility. In *HICSS'05 proceedings of the 38th IEEE annual Hawaii international conference on system sciences* (p. 313b).

Ruhe, G., & Saliu, M. O. (2005). The art and science of software release planning. *IEEE Software*, *22*(6), 47–53.

Saliu, O., & Ruhe, G. (2005). Software release planning for evolving systems. *Innovations in Systems and Software Engineering*, *1*(2), 189–204.

Sukhwani, H., Alonso, J., Trivedi, K. S., & Mcginnis, I. (2016). Software reliability analysis of NASA space flight software: A practical experience. In *2016 IEEE international conference on software quality, reliability and security* (pp. 386–397).

Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S. B., & Shafique, M. U. (2010). A systematic review on strategic release planning models. *Information and Software Technology*, *52*(3), 237–248.

Szöke, Á. (2011). Conceptual scheduling model and optimized release scheduling for agile environments. *Information and Software Technology*, *53*(6), 574–591.

Tsay, J., Wright, H. K., & Perry, D. E. (2011). Experiences mining open source release histories. In *Proceedings of the ACM 2011 international conference on software and systems process* (pp. 208–212).

Yamada, S., Hishitani, J., & Osaki, S. (1993). Software-reliability growth with a Weibull test-effort: A model and application. *IEEE Transactions on Reliability*, *42*(1), 100–106.

Yamada, S., Ohtera, H., & Narihisa, H. (1986). Software reliability growth models with testing-effort. *IEEE Transactions on Reliability*, *35*(1), 19–23.

Yang, J., Liu, Y., Xie, M., & Zhao, M. (2016). Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes. *Journal of Systems and Software*, *115*, 102–110.

Zhu, M., & Pham, H. (2016). A software reliability model with time-dependent fault detection and fault removal. *Vietnam Journal of Computer Science*, *3*(2), 71–79.

Zhu, M., Zhang, X., & Pham, H. (2015). A comparison analysis of environmental factors affecting software reliability. *Journal of Systems and Software*, *109*, 150–160.